Aeonlink

A Framework for AI-Agent Platforms

Agents, policy, pricing and applications as configuration over a single runtime

Marcel Langjahr

Pantareon
Pantareon Foundations

langjahr@pantareon.io · https://pantareon.io/

July 2026

Abstract

Aeonlink is a framework for building and operating AI-agent platforms: the complete horizontal stack — identity, memory, retrieval, moderation, metering, social surface and an application model — beneath whatever a company actually wants to offer its users. This paper describes that stack. The applications running on Pantareon's own instance are outside its scope; they are products built *with* the framework, not part of it.

The architecture rests on one decision: *the behaviour of a platform is defined by documents, not by code.* An agent is a document, injected verbatim into the model's context. A retrieval policy, a tool manifest, the whitelist of user-state fields a given conversation mode may see, the model registry with its per-token prices, an application's store listing and its entire AI behaviour, the moderation rulebook, the credit packages and the permission ladder are all documents. The runtime that reads them is generic and does not encode which platform it is serving. Adding an application, publishing an agent, opening a knowledge domain, repricing a model or changing moderation policy is a data change, not a deployment.

Five planes sit on that contract. A **runtime** in which one hardened model client serves agent conversation, in-application turns and platform moderation alike, with a retrieval router that decides what to look up before the agent ever speaks. A **memory** of three layers — encrypted transcript, distilled episodic shards over a similarity graph, and a periodically recalibrated long-term field — in which only retained conversation imprints and inference writes nothing. An **extension model** in which an application docks as a directory entry, receives a multiplayer session layer, an opaque per-user store and a directive-driven AI turn engine, and may be hosted internally or framed externally behind a single-sign-on handoff. An **economy** in which every model call is metered once, priced per model, and settled against a prepaid credit balance with a Stripe onramp. And a **social surface** of groups, direct messages, feeds, editable 3D worlds and quests, with server-authoritative games and placement-only ratings beneath it.

The implementation is 44,658 lines of Node/Express across 142 modules, exposing 330 endpoints through 33 route factories over 72 services and roughly 70 Firestore collections, with a 50,754-line React client including a self-contained 3D world engine. It runs as a single container on Cloud Run with secrets from a managed vault, a pure-logic self-test suite gating every image build, and a live deployment carrying real users, real agents and real money.

Scope of this Paper: This is a technical architecture overview for partners and prospective licensees. It describes the framework's mechanisms as they are implemented, with figures measured against the shipping repositories. It is not a product brochure and not a research paper. Applications that ship on Pantareon's own instance — games, tutors, research tools, editorial and travel products — appear only where they illustrate an extension point; they are not part of what this document describes and would not be part of a framework licence.

1 Introduction

A company that wants an AI-agent product of its own faces a choice that is usually posed badly.

It can build the stack. That means authentication and session management, a billing system that meters model usage accurately enough to charge for it, a memory architecture that survives beyond a context window, a retrieval layer with permissions,

a moderation pipeline that holds up legally, a social surface, and an application model — perhaps a year of undifferentiated engineering before the first feature that is genuinely theirs.

Or it can rent a model API and discover that everything distinctive about an agent platform is precisely what the API does not provide. The model is a component, and a good one; it is not a platform. Who the agents are, what they remember across months, what they are permitted to look up, how they are moderated, what a conversation costs and who pays for it — none of that arrives with the tokens.

Aeonlink exists because most of that work is horizontal. It is the same for an agent platform in education, in retail support, in entertainment or in an enterprise's internal tooling. It can be built once, well, and delivered as a foundation.

The organising decision is what makes that foundation adaptable rather than merely reusable. **The behaviour of the platform is defined by a set of documents, and the engine that reads them does not know which platform it is.** An agent is a document. A retrieval policy is a document. The model registry and its prices, an application's launch contract and its whole AI behaviour, the moderation rulebook, the credit packages, the permission ladder and the per-mode context whitelists are documents. The runtime loads them, caches them briefly, and runs.

The practical consequence is measured in what a change costs. On the running platform, flipping one field in an application's document makes it appear in the application grid, in the arcade and in the discovery surface, with no code edit anywhere. Opening a knowledge domain, publishing a new agent, re-pointing the default model after a vendor deprecation, adjusting a price or rewriting the moderation policy are all writes to a database. *A licensee adapts the platform by editing documents.*

Three capabilities follow from that construction rather than being engineered on top of it.

One runtime serves every consumer. Agent conversation, in-application AI turns and platform-side moderation compose the same model client, the same retry and normalisation layer, the same context-whitelist mechanism and the same billing choke point. A turn is assembled, sent, metered and persisted through one pipeline regardless of who is speaking.

Only retained conversation imprints. The dialogue model has no write path to any memory store. Memories form from the transcript the user chose to keep, and the long-term field forms from those memories. This is a structural property of the pipeline, not a policy applied to it.

Every model call passes one meter. There is no unmetered inference on any user-facing path. Token usage across dialogue, memory formation and embedding is accumulated and settled exactly once per turn against a prepaid balance. The economy is not an add-on; it is a stage every turn is routed through.

2 Architecture

The backend is a single Node/Express service with one composition root. That root — 724 lines — is the only place wiring happens: it initialises Firebase Admin, assembles the middleware chain, loads secrets from a managed vault, constructs roughly thirty services with explicit hand-wired dependencies, builds the rate limiters and upload handling, and mounts thirty-three route factories. Every route module is a function of one dependency bag and returns a router. There is no service locator, no ambient global state and no framework magic; the entire object graph of the application is readable in one file.

Persistence is Firestore throughout, with blob storage in a bucket and secrets in a managed vault. A central error handler guarantees that every failure path, including body-parse and upload errors, returns the same JSON shape.

Five planes sit on that root.

The configuration plane holds everything an operator tunes: documents under a settings collection, the universal interaction rules split into editable sections, and agent directives in their own collection. Section 3 describes it, because the rest of the framework is built to be driven by it.

The runtime plane is one model client behind one retry-and-normalise wrapper, a retrieval router, an ordered prompt assembler, and a billing stage. Agent conversation, application turns and moderation each compose it differently.

The memory plane maintains, per user and agent, an encrypted transcript, a growing set of distilled shards linked by semantic similarity, and a single recalibrated long-term field.

The extension plane turns an application into a document. Internal applications additionally receive a session and lobby layer, an opaque per-user key–value store, and a compiler that renders a directive document into a structured prompt. External applications are framed with a single-sign-on hand-off.

The interaction plane is the social and game surface: groups, direct messages, feeds, editable

worlds, quests, and a server-authoritative game substrate with ratings.

3 The Configuration Plane

The framework's adaptability rests on a single invariant: *what the platform is gets changed by editing documents, and the engine stays untouched.*

An agent is a document. A persona directive carries a core directive and any number of additional fields — voice, few-shot examples, boundaries. It is loaded once per turn and injected verbatim as a late part of the prompt. The same loader serves one-to-one conversation, group conversation and in-application turns, so an agent behaves consistently wherever it appears. Adding an agent to the platform is writing a document and listing its name in a registry.

Retrieval policy is a document. The router's directive, the manifest of retrieval tools available to it, and the mapping of which agents may access which knowledge domains are all configuration. Access is default-deny and evaluated server-side at execution time, not merely suggested to the model.

Context exposure is a document. For each conversation mode the framework keeps an explicit whitelist of user-state fields permitted to enter the model's context. Widening or narrowing what the model can see about a user is a configuration change with a reviewable diff — a property that matters when a data-protection officer asks what leaves the database.

The model registry is a document. Model identifiers and their per-million-token input and output prices live in configuration and are cached briefly in process. When a vendor deprecates a model line, migration is one field. Prices used for billing are read from the same place, so metering and model selection can never drift apart.

An application is a document. One entry carries the store listing — name, descriptions, icon, status — and the launch contract: internal or external, its domain, its embedding mode, its hand-off behaviour. A separate directive document carries the application's entire AI behaviour as ordered, labelled sections that the framework compiles into a structured prompt with per-call output schemas.

Policy and commerce are documents. The moderation rulebook, the permission ladder, the credit packages, the knowledge-domain registry and the platform's operational settings are all stored, versioned and editable without a deployment.

The engineering pattern beneath all of this is consistent: a typed reader with a short in-process cache, a code-level default where one is safe, and validation at write time. It is deliberately unglamorous, and it is what turns a platform into something a second party can shape.

4 The Runtime

An agent turn follows a fixed pipeline, and each stage exists for a reason worth stating.

An authenticated request reaches a thin route handler. A pre-flight balance check rejects the turn before any model call if the user cannot cover a minimum turn. Counter-triggered memory upkeep may run — forming a shard, or recalibrating the long-term field — *before* the dialogue, so the turn benefits from it. Then, in one-to-one conversation, the router runs.

The router is a separate, low-temperature, JSON-only model call whose entire job is to decide what to retrieve, and which never addresses the user. It sees the available knowledge domains, the conversation's recent topics, a compact window of the last exchanges so that elliptical follow-ups resolve, and the user's message. It emits knowledge queries with optional domain scoping and web queries. Separating this decision from the agent's voice keeps retrieval reasoning out of the character's mouth and lets it run cheaply and deterministically.

Its results are executed with two guarantees that matter more than they look. A query against a domain the agent may not access is dropped and *replaced with an explicit marker* telling the model that access was denied. A retrieval that returns nothing injects an equally explicit marker instructing the model not to invent an answer. Empty retrieval is the moment a language model is most likely to confabulate, and the framework makes that moment visible to it rather than silent.

Prompt assembly is ordered deliberately. Stable content — the framework's own interaction rules, the conversation transcript — goes first, so that it forms a reusable prefix for the provider's implicit caching. Volatile content — retrieval results, current state, the temporal anchor, the agent directive, the user's message — goes last. The temporal anchor is placed late on purpose: it changes every minute and would otherwise invalidate everything behind it.

That anchor is itself a small, careful subsystem. It resolves the user's timezone through a chain — live location if shared, home coordinates otherwise, operator default as the floor — using an offline lookup, and it states the current time to the model explicitly. Location enters the prompt only in one-to-one

conversation and only with opt-in; coordinates are used to derive a timezone and are never themselves placed in context.

The dialogue call then runs through a single hardened wrapper providing exponential backoff with jitter, error classification, and parameter normalisation that guards against provider SDK differences. One billing call settles the entire turn — dialogue, memory formation and embedding tokens together. Both messages are appended to the encrypted transcript with the resolved timezone recorded.

The same primitives are exported and composed by the application orchestrator for in-application turns, and a self-contained moderation service composes the client and whitelist mechanism separately. One runtime, three consumers, one place to harden.

5 Memory

Memory is the capability that distinguishes an agent from a chatbot, and the framework implements it in three layers, keyed per user and agent.

Working context is the transcript: an encrypted, transactionally sequenced, topic-tagged message log. Each turn loads the most recent window, tunable per user between 20 and 250 messages. Encryption is field-level AES-256-GCM with the key held in the managed vault.

Episodic memory is a set of distilled shards. Every few user turns, a cheap model pass reads the recent window and writes a single first-person memory — or explicitly declines, which is the more common and more valuable outcome, because a memory system that records everything remembers nothing. The shard is encrypted, embedded at 1536 dimensions, stored append-only, and linked into a similarity graph against existing shards.

Recall is a three-part blend under a fixed token budget: the most recent shards chronologically, a semantic top-k scored as 0.9 cosine similarity plus 0.1 linear recency, and associative expansion across the similarity graph so that a retrieved memory can pull in what it is connected to. The budget is the important part — memory is bounded by design, so its cost per turn is predictable.

The long-term field is a single condensed text per agent, seeded at creation from a birth signature and periodically rewritten in full by a calibration pass that reads the previous field together with every shard formed since the last one. This is the layer that gives an agent a sense of who someone is rather than a list of things they said.

The structural property worth stating precisely: *the dialogue call has no write path to any memory store.* Shards form from the retained transcript; the field forms from shards. A conversation the user deletes leaves nothing behind, and inference alone changes nothing durable. Memory is a consequence of retention, and the pipeline enforces that rather than asking the model to respect it.

All memory work is metered. Formation and embedding tokens are accumulated into the same per-turn usage object as the dialogue and settled in the single deduction, so a platform operator can see what memory actually costs.

Groups carry a parallel long-term field, recalibrated from recent group conversation and billed to the group's owner.

6 Agents

An agent is defined entirely by data: a persona directive document, an optional birth memory, and catalogue entries for its display names and descriptions. The user's relationship with it is one session document that carries everything personal to that pairing — a custom name assigned at creation from a deterministic hash-derived generator, the recalibrated long-term field, conversation topics, history preferences, progression counters, an optional per-agent model override, and a custom avatar.

Portability is genuine. The same directive loader and the same memory assembly run the agent in one-to-one conversation, inside a group, and within an application, where it reads its long-term memory but does not write to it — one writer, many readers. The active agent travels in the session token, and switching mints a new one.

On top sits a derivation layer that computes rather than stores. An archetype is derived from a hash of the identity; a character sheet is computed on demand from accumulated progression and that archetype; names are generated deterministically from the same source. Nothing needs migrating when the derivation changes, and every function in the chain is pure and independently testable.

For populated worlds, non-player agents are assembled from curated prose templates plus owner-supplied content, held inside a hard framing instruction that separates the author's intent from injectable text. Reward grants are proposed by the model as a marker and authorised by the engine — the model suggests, the server decides. That division is a pattern the framework applies wherever a model's output would otherwise have authority.

7 The Extension Model

An application docks onto the platform through one collection entry that serves simultaneously as its store listing and its launch contract. A public directory endpoint projects those entries for the client, so an application's presence, description and launch behaviour are data.

Internal applications receive four facilities from the framework.

A session and lobby layer. Six-character join codes, transactional seat claiming, host-controlled start, and a monotonic turn counter guarded by optimistic concurrency so that two intents cannot interleave. Server-authoritative games plug in through a registry that maps an application identifier to an engine module, so the route and ranking layers never branch on which game is running.

An opaque per-user store. A size-capped key–value document per user and application, scoped by the session token, validated against the application registry, and swept automatically on account erasure. An application persists user state with no backend change and inherits data-rights compliance for free.

A directive-driven turn engine. An application's AI behaviour is a document whose ordered sections the framework compiles into labelled prompt blocks, with per-call output schemas and behaviour flags controlling history, tool use and structured output. A new AI-driven application is, in the common case, a settings document.

Metering and quotas. Application turns run through the same billing stage as everything else, and a quota pattern provides free-tier allowances with daily rollover and duplicate-turn protection.

External applications — self-hosted pages that should appear inside the platform — are framed by one generic component with a single-sign-on handoff: the frame announces readiness, the host replies with the session token targeted at an exact origin, and both sides validate. Transactional email sent on behalf of a framed application carries that application's branding, resolved from the same directory document, with the redirect target validated server-side.

8 Knowledge and Moderation

The knowledge base is a single collection in which the domain is a validated field rather than a path, so one vector space spans the corpus and domain scoping is a filter over it.

Ingestion is deliberately two-phase. An AI pass proposes metadata — title, summary, tags, domain — and, for oversized input, produces a condensation. A separate deterministic commit then validates the proposal, checks slug ownership, *embeds before writing anything*, stores the original document in blob storage, and only then writes the record. Because embedding precedes persistence, a failed embedding leaves no half-ingested document behind, and every stored record is retrievable by construction.

Retrieval enters a conversation at exactly one point — the router — under per-agent domain permissions evaluated server-side. Results are scored by cosine similarity above a floor, with an optional community-signal prior and a per-domain recency preference for time-sensitive material.

The lifecycle around the corpus is complete rather than aspirational. Contribution is role-gated and fails closed. Readers can star documents. Any user can report one. A moderation rejection quarantines the document — removed from search and listings while the record and its original are retained for audit — and privileged roles can restore or permanently delete it. Every decision is logged.

Moderation itself is a policy-driven judge with a mechanism worth describing, because it addresses the usual weakness of AI moderation in publishing flows. An approval is bound by content hash to the exact material approved and is valid only for a short window; the server loads the content itself rather than accepting it from the client. A client therefore cannot obtain approval for benign text and publish something else. Reports carry an economic cost borne by whichever side is wrong, which prices out mass reporting, and removals carry a formal notice.

9 The Economy

The framework meters and bills model usage as a first-class concern, because an agent platform that cannot attribute cost to conversation cannot be operated commercially.

The unit of account is a prepaid credit pegged to a hundredth of a euro. Credits are purchased through hosted checkout against server-resolved packages, with a signature-verified webhook, an idempotency marker per payment, and consent capture before purchase. Tax handling is integrated and configurable.

per 1M tokens	economy	premium
input	$0.25	$2.00
output	$1.50	$12.00
embedding		$0.15

Table 1: Model prices as held in the registry. Credit value, currency conversion and platform margin sit beside them as configuration and apply on top.

Three billing primitives cover the platform's needs: metered post-hoc deduction for AI turns, an all-or-nothing flat charge with refund support for bounded jobs, and a voluntary forfeiture path. Every movement lands in a single transaction ledger that a user can inspect.

The metered path is the load-bearing one. Token counts from the provider response are priced against the resolved model, converted, marked up, and deducted transactionally. Each AI route gates on a minimum-turn floor before calling the model and clamps the deduction at a zero balance, so a turn is either afforded or refused rather than half-delivered. Free tiers ride a per-application quota document with daily allowances and duplicate-turn protection.

Group agents introduce a second payer: a group's agent answers on request at the requester's expense, or automatically at the owner's, under a cooldown and a transaction lock, pausing itself when the owner's balance runs out and resuming when it is topped up.

10 The Interaction Surface

One collection implements groups, direct messages, per-user feeds, public 3D worlds and a quest layer, unified so that membership, moderation, media handling and AI participation are written once.

Messages are encrypted at rest under the platform key. Media is handled by reference rather than by URL: a message points at a document in the sender's own workspace, and bytes are streamed through access-checked routes that verify membership or ownership on every request. Voice notes are transcoded and transcribed; images are downscaled, thumbnailed and described; documents are text-extracted and summarised. Each pipeline is metered.

Public groups can carry an editable 3D world validated against a generated whitelist of permitted props and grids, an owner-built graph of sub-locations, and non-player agents with server-authorised progression. Presence is a heartbeat with warm and cold tiers. Feeds publish only against a fresh, hash-bound moderation approval.

The transport is incremental REST polling on a tiered rate limit, with push notifications used to wake clients. This is a deliberate simplicity choice: it keeps the server stateless, survives instance recycling without reconnection logic, and made the platform straightforward to operate on a scale-to-zero runtime.

Beneath the sessions sits a game substrate that is the framework's cleanest generic design. An engine implements a narrow contract — initial state, apply intent, apply forfeit, list legal intents, terminal test, final standings, per-seat redaction. The server rolls every die under a commit-reveal scheme and is the sole writer of game state; each applied intent advances a counter under optimistic concurrency. AI seats are driven server-side by an in-memory runner with injected decision providers, protected by a lease so that a slow model call cannot corrupt a concurrent human move.

When a match ends, one transaction writes an immutable result ledger, updates placement-only Glicko-2 ratings through a game-agnostic rating engine, refreshes a conservative-score leaderboard, updates per-subject statistics and settles progression rewards — made idempotent by a recorded flag, and downgraded to personal statistics when too few distinct humans took part for a rating to mean anything. Score magnitude never enters the rating; only placement does.

11 Identity, Privacy and Data Rights

Identity is email and password with verification, reset, and stateless session tokens carrying a version claim, so that a password change invalidates outstanding sessions. Rate limiting on authentication is backed by a shared store so it holds across instances of a horizontally scaled service.

Sensitive fields are encrypted at rest with a key from the managed vault. Uploads are validated by content sniffing rather than by declared type. Blob access outside chat runs through short-lived signed URLs; chat media, as described, never leaves the access-checked path.

The framework's strongest privacy mechanism is structural. **One declarative map enumerates every location where user data lives**, and both the data-subject export and the erasure path are driven from that single map rather than from two independently maintained lists. Adding a collection means adding one entry, and the export and deletion follow automatically. The map additionally records retention obligations, so data that must be kept for

statutory reasons is distinguished from data that must be erased — and the map itself is covered by an automated test.

Consent is versioned and re-gated when the version changes, with the accepted version and timestamp stored as proof. Public discovery is tiered: private, platform-visible and public, with identity-stripped projections for unauthenticated surfaces.

12 Engineering and Operations

quantity	measured
backend	44,658 LOC, 142 modules
composition root	724 LOC
frontend (JS/JSX)	50,754 LOC
endpoints	330 in 33 routers
services	72
collections	≈70
dependencies	22
self-tests	21 (≈660 checks)

Table 2: Measured against the shipping repositories.

The backend runs as a single container on Cloud Run in a European region, with a warm instance held to avoid cold starts on user-facing latency. Secrets load at boot from a managed vault, with required and optional tiers so that a missing peripheral integration degrades a feature rather than preventing startup. Shutdown drains connections on signal. The client is a React single-page application built with Vite and published to managed hosting, carrying a build-identity heartbeat that reloads stale tabs after a deployment and a service worker whose navigation strategy is network-first for exactly that reason.

Verification is a hand-rolled, dependency-free self-test harness: 21 pure-logic modules carrying roughly 660 assertions, run sequentially by a small runner with an exit-code contract, gating the container build. Because the suite touches no datastore, network or environment, it runs in 3.3 seconds and can be executed anywhere, including inside a build step or on a developer machine with no credentials. It covers the deterministic cores — game rules against a golden master with fuzzing, rating mathematics, payload validation, ranking settlement against a datastore mock, temporal resolution, retry classification and the user-data map.

The frontend contributes a substantial capability of its own: a self-contained 3D world engine built directly on Three.js, rendering declarative world definitions with an auto-discovered prop registry, alongside a dual-renderer avatar kit in which a figure is stored as named part slots and palette roles and realised independently in 2D vector and 3D mesh form, whitelist-sanitised on both client and server.

13 What a Licence Conveys

Concretely, a licensee receives the framework described above:

- **The runtime** — one hardened model client, the retrieval router with its anti-confabulation guarantees, ordered prompt assembly with cache-aware structure, the temporal and location anchor, and the per-mode context whitelist.
- **The memory architecture** — encrypted transcript, distilled shards over a similarity graph with bounded recall, and the recalibrated long-term field, with retention-only imprinting and full cost attribution.
- **The agent system** — agents as documents, portable across conversation, groups and applications, with deterministic derivation of names, archetypes and progression.
- **The extension model** — applications as directory entries, the session and lobby layer, the per-user store, the directive-to-prompt compiler, and the external-application hand-off with white-labelled transactional email.
- **Knowledge and moderation** — two-phase ingestion, permissioned domains, the full contribution-to-quarantine lifecycle, and hash-bound approval for publishing.
- **The economy** — prepaid credits, per-model metering, hosted checkout with idempotent settlement, quota-based free tiers, and a full transaction ledger.
- **The interaction surface** — groups, direct messages, feeds, editable worlds, quests, media pipelines, and the server-authoritative game substrate with placement-only ratings.
- **Identity and data rights** — session management with revocation, encryption at rest, and the declarative data map driving export and erasure.
- **The client** — the application shell, the 3D world engine and the avatar kit.

What a licence does not convey is the products built on the framework. The applications running on

Pantareon's instance, their agents and their fictional setting are our own; they are examples of what the framework carries, not part of it.

14 Conclusion

Aeonlink starts from the observation that an AI-agent platform is mostly horizontal engineering, and that the part which is not — who the agents are, what they may retrieve, how they are moderated, what a conversation costs — is better expressed as configuration than as code.

Built that way, the platform's distinguishing content becomes a document set over a runtime that does not need to know which platform it serves. Agents, retrieval policy, model prices, application manifests and moderation rules are data. One runtime serves agent conversation, application turns and moderation alike. Only retained conversation imprints on memory. Every model call passes exactly one meter.

That is a foundation a company can adopt rather than rebuild, and it is running today with real users, real agents and real money moving through it.